Damien Chiem Exercise 2

MAT 115

Exercise #2

As before, we load the `dslabs` package first.

```
library(dslabs)
```

Section 2.4 is mostly about dataframes. Take a look at dataframes that are included in the `dslabs` package by using the following command:

```
data(package="dslabs")
```

Pick one of the dataframes and look at it. (R is not very visual. It can be a bit disorienting if you are used to looking at spreadsheets in Excel or Sheets.) There are many ways to take a look: you can just type the name of the dataset, you can use the `View()` command (note the capital V, R is case sensitive), you can use the `head` command (shows the first 6 rows of a dataframe), and there are others.

```
death_prob #2015 US Period Life Table
```

```
##     age    sex     prob
## 1     0   Male 0.006383
## 2     1   Male 0.000453
## 3     2   Male 0.000282
## 4     3   Male 0.000230
## 5     4   Male 0.000169
## 6     5   Male 0.000155
## 7     6   Male 0.000145
## 8     7   Male 0.000135
## 9     8   Male 0.000120
## 10    9   Male 0.000105
## 11   10   Male 0.000094
## 12   11   Male 0.000099
## 13   12   Male 0.000134
## 14   13   Male 0.000207
## 15   14   Male 0.000309
## 16   15   Male 0.000419
## 17   16   Male 0.000530
## 18   17   Male 0.000655
## 19   18   Male 0.000791
## 20   19   Male 0.000934
## 21   20   Male 0.001085
## 22   21   Male 0.001228
## 23   22   Male 0.001339
## 24   23   Male 0.001403
## 25   24   Male 0.001433
## 26   25   Male 0.001451
## 27   26   Male 0.001475
## 28   27   Male 0.001502
## 29   28   Male 0.001538
## 30   29   Male 0.001581
## 31   30   Male 0.001626
## 32   31   Male 0.001669
## 33   32   Male 0.001712
## 34   33   Male 0.001755
## 35   34   Male 0.001800
## 36   35   Male 0.001855
## 37   36   Male 0.001920
## 38   37   Male 0.001988
## 39   38   Male 0.002060
## 40   39   Male 0.002141
## 41   40   Male 0.002240
## 42   41   Male 0.002362
## 43   42   Male 0.002509
## 44   43   Male 0.002684
## 45   44   Male 0.002890
## 46   45   Male 0.003121
## 47   46   Male 0.003386
## 48   47   Male 0.003707
## 49   48   Male 0.004091
## 50   49   Male 0.004531
## 51   50   Male 0.005013
## 52   51   Male 0.005524
## 53   52   Male 0.006059
## 54   53   Male 0.006611
## 55   54   Male 0.007187
## 56   55   Male 0.007800
## 57   56   Male 0.008456
## 58   57   Male 0.009144
## 59   58   Male 0.009865
## 60   59   Male 0.010622
## 61   60   Male 0.011458
## 62   61   Male 0.012350
## 63   62   Male 0.013235
## 64   63   Male 0.014097
## 65   64   Male 0.014979
## 66   65   Male 0.015967
## 67   66   Male 0.017109
## 68   67   Male 0.018392
## 69   68   Male 0.019836
## 70   69   Male 0.021465
## 71   70   Male 0.023351
## 72   71   Male 0.025482
## 73   72   Male 0.027794
## 74   73   Male 0.030282
## 75   74   Male 0.033022
## 76   75   Male 0.036201
## 77   76   Male 0.039858
## 78   77   Male 0.043891
## 79   78   Male 0.048311
## 80   79   Male 0.053228
## 81   80   Male 0.058897
## 82   81   Male 0.065365
## 83   82   Male 0.072491
## 84   83   Male 0.080288
## 85   84   Male 0.088916
## 86   85   Male 0.098576
## 87   86   Male 0.109438
## 88   87   Male 0.121619
## 89   88   Male 0.135176
## 90   89   Male 0.150109
## 91   90   Male 0.166397
## 92   91   Male 0.183997
## 93   92   Male 0.202855
## 94   93   Male 0.222911
## 95   94   Male 0.244094
## 96   95   Male 0.265091
## 97   96   Male 0.285508
## 98   97   Male 0.304926
## 99   98   Male 0.322919
## 100  99   Male 0.339065
## 101 100   Male 0.356018
## 102 101   Male 0.373819
## 103 102   Male 0.392510
## 104 103   Male 0.412135
## 105 104   Male 0.432742
## 106 105   Male 0.454379
## 107 106   Male 0.477098
## 108 107   Male 0.500953
## 109 108   Male 0.526000
## 110 109   Male 0.552300
## 111 110   Male 0.579915
## 112 111   Male 0.608911
## 113 112   Male 0.639357
## 114 113   Male 0.671325
## 115 114   Male 0.704891
## 116 115   Male 0.740135
## 117 116   Male 0.777142
## 118 117   Male 0.815999
## 119 118   Male 0.856799
## 120 119   Male 0.899639
## 121   0 Female 0.005374
## 122   1 Female 0.000353
## 123   2 Female 0.000231
## 124   3 Female 0.000165
## 125   4 Female 0.000129
## 126   5 Female 0.000116
## 127   6 Female 0.000107
## 128   7 Female 0.000101
## 129   8 Female 0.000096
## 130   9 Female 0.000092
## 131  10 Female 0.000091
## 132  11 Female 0.000096
## 133  12 Female 0.000111
## 134  13 Female 0.000138
## 135  14 Female 0.000174
## 136  15 Female 0.000214
## 137  16 Female 0.000254
## 138  17 Female 0.000294
## 139  18 Female 0.000330
## 140  19 Female 0.000364
## 141  20 Female 0.000399
## 142  21 Female 0.000436
## 143  22 Female 0.000469
## 144  23 Female 0.000497
## 145  24 Female 0.000522
## 146  25 Female 0.000546
## 147  26 Female 0.000572
## 148  27 Female 0.000604
## 149  28 Female 0.000644
## 150  29 Female 0.000690
## 151  30 Female 0.000740
## 152  31 Female 0.000792
## 153  32 Female 0.000841
## 154  33 Female 0.000886
## 155  34 Female 0.000929
## 156  35 Female 0.000977
## 157  36 Female 0.001034
## 158  37 Female 0.001098
## 159  38 Female 0.001171
## 160  39 Female 0.001253
## 161  40 Female 0.001347
## 162  41 Female 0.001452
## 163  42 Female 0.001571
## 164  43 Female 0.001706
## 165  44 Female 0.001857
## 166  45 Female 0.002022
## 167  46 Female 0.002204
## 168  47 Female 0.002411
## 169  48 Female 0.002648
## 170  49 Female 0.002910
## 171  50 Female 0.003193
## 172  51 Female 0.003491
## 173  52 Female 0.003801
## 174  53 Female 0.004119
## 175  54 Female 0.004449
## 176  55 Female 0.004813
## 177  56 Female 0.005201
## 178  57 Female 0.005583
## 179  58 Female 0.005952
## 180  59 Female 0.006325
## 181  60 Female 0.006749
## 182  61 Female 0.007238
## 183  62 Female 0.007776
## 184  63 Female 0.008368
## 185  64 Female 0.009032
## 186  65 Female 0.009794
## 187  66 Female 0.010673
## 188  67 Female 0.011676
## 189  68 Female 0.012815
## 190  69 Female 0.014105
## 191  70 Female 0.015616
## 192  71 Female 0.017318
## 193  72 Female 0.019118
## 194  73 Female 0.020996
## 195  74 Female 0.023033
## 196  75 Female 0.025413
## 197  76 Female 0.028197
## 198  77 Female 0.031313
## 199  78 Female 0.034782
## 200  79 Female 0.038689
## 201  80 Female 0.043258
## 202  81 Female 0.048490
## 203  82 Female 0.054223
## 204  83 Female 0.060446
## 205  84 Female 0.067338
## 206  85 Female 0.075133
## 207  86 Female 0.084033
## 208  87 Female 0.094177
## 209  88 Female 0.105633
## 210  89 Female 0.118407
## 211  90 Female 0.132476
## 212  91 Female 0.147801
## 213  92 Female 0.164331
## 214  93 Female 0.182012
## 215  94 Female 0.200783
## 216  95 Female 0.219758
## 217  96 Female 0.238630
## 218  97 Female 0.257065
## 219  98 Female 0.274706
## 220  99 Female 0.291189
## 221 100 Female 0.308660
## 222 101 Female 0.327180
## 223 102 Female 0.346810
## 224 103 Female 0.367619
## 225 104 Female 0.389676
## 226 105 Female 0.413057
## 227 106 Female 0.437840
## 228 107 Female 0.464111
## 229 108 Female 0.491957
## 230 109 Female 0.521475
## 231 110 Female 0.552763
## 232 111 Female 0.585929
## 233 112 Female 0.621085
## 234 113 Female 0.658350
## 235 114 Female 0.697851
## 236 115 Female 0.739722
## 237 116 Female 0.777142
## 238 117 Female 0.815999
## 239 118 Female 0.856799
## 240 119 Female 0.899639
```

```
# Write the name of a dataframe between the parenthesis below in View,
# then remove the comment sign. Run the chunk.
View(death_prob)
```

```
# Write the name of a dataframe between the parenthesis below.
# Remove the comment sign.
head(death_prob)
```

```
##   age  sex     prob
## 1   0 Male 0.006383
## 2   1 Male 0.000453
## 3   2 Male 0.000282
## 4   3 Male 0.000230
## 5   4 Male 0.000169
## 6   5 Male 0.000155
```

At this point you probably want to use `View()` since it gives you the most familiar version of a dataframe. But others are useful too.

Section 2.4.3 introduces the very common operator `$`. We use it to refer to a variable (i.e., a column) in a dataframe. To be specific, let's take a look at the `movielens` dataframe. The following command tells you the names of the columns in the `movielens` dataframe:

```
names(movielens)
```

```
## [1] "movieId"   "title"     "year"      "genres"    "userId"    "rating"
## [7] "timestamp"
```

To refer to the column `rating`, we say `movielens$rating`. For example, to calculate the mean of the ratings, we write:

```
mean(movielens$rating)
```

```
## [1] 3.543608
```

Analogy: when we fill out an official form, we usually write the family name first, followed by the given name (for example: Howard, Aaron). The name of the dataframe corresponds to the family name, and the name of the variable corresponds to the given name. And `$` corresponds to comma separating the family name from the given name.

Now try to find the maximum rating of the movies. Try to find the minimum rating. You'll have to guess the appropriate commands.

```
max(movielens$rating)
```

```
## [1] 5
```

```
min(movielens$rating)
```

```
## [1] 0.5
```

The columns of a data frame are examples of *vectors*. See the starting discussion of section 2.4.4. There are unfortunately many types of vectors (numeric, character, logical, factor, and so on), and R treats each type differently. You can find out the type of a vector by using the `class()` command.

```
class(movielens$movieId) #--> integer
```

```
## [1] "integer"
```

Use the R chunk above to find out the type of all the variable in the data frame `movielens`.

Type your answer here.

```
class(movielens$movieId) #--> integer
```

```
## [1] "integer"
```

```
class(movielens$title) # --> character
```

```
## [1] "character"
```

```
class(movielens$year) # --> integer
```

```
## [1] "integer"
```

```
class(movielens$genre) # --> factor
```

```
## [1] "factor"
```

```
class(movielens$userId) # --> integer
```

```
## [1] "integer"
```

```
class(movielens$rating)# --> numeric
```

```
## [1] "numeric"
```

```
class(movielens$timestamp) # --> integer
```

```
## [1] "integer"
```

You can also use the `str()` command to obtain a summary of the data frame:

```
str(movielens)
```

```
## 'data.frame':    100004 obs. of  7 variables:
##  $ movieId  : int  31 1029 1061 1129 1172 1263 1287 1293 1339 1343 ...
##  $ title    : chr  "Dangerous Minds" "Dumbo" "Sleepers" "Escape from New York" ...
##  $ year     : int  1995 1941 1996 1981 1989 1978 1959 1982 1992 1991 ...
##  $ genres   : Factor w/ 901 levels "(no genres listed)",..: 762 510 899 120 762 836 81 762 844 899 .
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ rating   : num  2.5 3 3 2 4 2 2 2 3.5 2 ...
##  $ timestamp: int  1260759144 1260759179 1260759182 1260759185 1260759205 1260759151 1260759187 1260
```

Note: `str` does not mean `string`. It's actually an abbreviation for `structure`.

You can see that `movielens$genres` is a *factor* with 901 *levels*. Factors and its levels are important in statistical analysis, especially in Analysis of Variance, among others. For now, think of `factor` as another word for type or kind, and `level` as the actual type or kind.

For example, the type of movie might be `action` or `comedy`. Then `action` and `comedy` are two specific levels of the factor `genres`. To see the levels of a factor, you can use the `levels` command:

```r
levels(movielens$genres)
```

(When you run the code chunk above, you should see all 901 levels of `genres`. But when you knit to pdf, the `results=FALSE` option means that the list of 901 levels is hidden — printing out 901 items is just too much. Try it out!)

Let's take a look at a different dataset, say `stars` in the `dslabs` package.

```r
str(stars)
```

```
## 'data.frame':    96 obs. of  4 variables:
##  $ star     : Factor w/ 95 levels "*40EridaniA",..: 87 85 48 38 33 92 49 79 77 47 ...
##  $ magnitude: num  4.8 1.4 -3.1 -0.4 4.3 0.5 -0.6 -7.2 2.6 -5.7 ...
##  $ temp     : int  5840 9620 7400 4590 5840 9900 5150 12140 6580 3200 ...
##  $ type     : chr  "G" "A" "F" "K" ...
```

You can see that there are 4 variables: the name of the star, its magnitude (which is a measure of how bright the star is — lower magnitude stars are brighter (!), the temperature of the star, and its stellar classification.

You can also see that the name of the star is misclassfied. It's not a `factor`; it should be `character`. (Why?) You can fix this mistake by reclassfying the `star` variable:

```r
stars$star <- as.character(stars$star)
class(stars$star)
```

```
## [1] "character"
```

You should also note that in the dataset, the variable `type` is classified as `character`. But I think it is more appropriate to classify it as `factor` since, after all, `type` tells us what type of star it is. You should change the classification of `type` into `factor`.

```r
# Write your command here.
# Then check that all four variables are now correctly classified.
stars$type <- as.factor(stars$type)
class(stars$star)
```

```
## [1] "character"
```

```
class(stars$magnitude)
```

```
## [1] "numeric"
```

```
class(stars$temp)
```

```
## [1] "integer"
```

```
class(stars$type)
```

```
## [1] "factor"
```

When you give the command `levels(some factor)` you get a list of the levels of the factor, usually arranged alphabetically. Sometimes you want the list to be ordered by a different criterion. For example, the type of stars is in part based on its temperature. Type O is the hottest stars, type M is the coolest stars. So you might want to ask R to list the levels of `type` based on temperature, not based on the alphabet.

```
stars$type <- reorder(stars$type, stars$temp, FUN=mean)
# now ask R to show you the levels of the factor `type`.
# Does R list them from hottest to coolest or from coolest to hottest?
levels(stars$type)
```

```
##  [1] "M"  "K"  "G"  "F"  "A"  "DF" "DA" "DB" "B"  "O"
```

```
# R lists them from coolest to hottest
```

Having the levels listed in reasonable order is important later when we make visual displays (and in analyzing data too).

Section 2.4.6 is about lists. You may know that lists are one of the fundamental objects in programming languages. We won't use lists too much in this course, since we will mostly work with dataframes, but when you clean up data, you will often have to manipulate lists, so it is important to know how R handles lists. Take a look at section 2.4.6. I don't have much to add to this section. But you should know that a variable in a dataframe is a special kind of list, and the list manipulation commands can be used on it.

For example, if you want to find out what star is in the fifteenth position in the `stars` dataframe, you can use the following:

```
stars$star[[15]]
```

```
## [1] "Spica"
```

Note: You need `[[` when working with lists. This is because when `[` is applied to a list it always returns a list. It never gives you the contents of the list. To get the contents, you need `[[`.

Finally, section 2.4.7 is about matrices. A matrix in R has to consist of data of the same type, usually numbers. So a dataframe is not a matrix. But some of the commands for matrices will work on dataframes. For example, to take a look at the 10th row of the `stars` dataframe, you can issue the following command, which uses brackets ('[ ]') for indexing.

```
stars[10,]
```

```
##          star magnitude temp type
## 10 Betelgeuse      -5.7 3200    M
```

This tells you that Betelgeuse is a type M star, with magnitude -5.7 and 3200 degree temperature.

```
# Take a look at all the stars' temperatures using a matrix command.
# Can you find the mean temperature of all the stars in this dataset?
# (You can of course use `mean(stars$temp)`, but try using the square bracket).

stars[,3]
```

```
##  [1]  5840  9620  7400  4590  5840  9900  5150 12140  6580  3200 20500 25500
## [13]  8060  4130 25500  3340  9060  4900  9340 28000 13260 28000 23000 25500
## [25] 23000  9620  3750 25500  4730 15550  9300 12400 26950  7700 33600  4900
## [37]  9900 20500 11000  7400  4590 20500 20500  4900  6100  9900  9340  6100
## [49] 25500  9900  2670  4900  2800  2670  3200  2670  2670  9620 14800  2800
## [61]  2670  4590  2800  2670  3340  2670  4130  4130  3870  3070  2940  5150
## [73]  6600  9700  3340  2500  2670  2800  3340  3480  2940  2670  2940  2800
## [85]  2670 13000  2800  3200  8060  2940  4900 10000  2940  4950  3870  2800
```

```
mean(stars[,3])
```

```
## [1] 8752.292
```